Udemy

Java Virtual Threads & Concurrency Masterclass [Hands-On] -by vinoth sel -current

multithreading-and-parallel-computing-in-java/learn – wonderful course – contains everything –every fork join pool

modern-java-multithreading-in-java-using-virtual-threads – By pragmatic

Ref links

|  |  |
| --- | --- |
| sir git link | <https://github.com/vinsguru/java-virtual-thread-course> |
|  |  |

Terminologies

1. Platform thread = os thread

IO intensive task – where the processor regularly wait for Input/Output – like reading from disk, writing to a disk, network REST svc calls, interacting with db

Anything which is slower than main memory, where reading is slow like network calls, db calls

Thread is the one who processes the incoming request

Ex;- if our backend service is hit by users, to process those req, tomcat will have server threads those will handle that incoming requests

1. Demon threads are the helper threads- if main threads dies- then app will exit & it wont wait for helper threads to finish their task

Ex:- if any day if hero & heroin and director are not present for the shoot, then even if other persons present they will cancel the shoot

Ex:- garbage collector thread is daemon thread

Important points

1. Thread is just like a postman, actual thread method code / all lines will be executed by processor , scheduler is the one who allocates cpu to the thread
2. We cant create millions of threads - 1 java platform thread = 1 os thread = so in real time that many os threads will not at all be available, for every thread os has to allocate 1-2MB stack mem, so in real time that much mem also will not be available
3. Creating Thousands of threads doesn’t give parallalism = bec single core proce is present = threads will be idle and will just be waiting for cpu allocation all the time

So even if u create 1000 threads if single proc is there then only 1 thread will be give chance at a time (rem 999 threads will be ideal)

In real time processor is the one which executes all our instructions /each line of code, threads will just bring the lines to the processor

1. Every thread has is own stack area (1 MB)- to store local var ref and to store which method it is currently exec

|  |  |
| --- | --- |
| Process  An instance of computer program It includes code,resources(memory allocated by os ), & it is heavy weight, It is very expensive to create and destroy a process    entire app is considered as a process (MS word , sumatra pdf apps are called processes)  inside app we have threads | **Thread**    Thread is a part of process (A process contain one or more threads)  above pic says Threads with in a process can share the memory and space |
| processor    cpu == processor both are same  scheduler is the one which will allocates cpu to a thread  cpu is like a bike , at any time only 1 can travel  1 cpu can execute 1 thread at a time  ex:- here even though 3 threads are there, since only 1 cpu is present, at any time only 1 thread will be executed, 1st 5 mins t1 and then t2 & then t3 will be given chance  doubt- if 1 cpu is present then whats use of creating multiple threads as at any time only 1 thread will be given chance | **multi core processor**    if 2 cpu are there, each cpu can exec 1 thread at any point of time |

Analogies

1. A thread = a server in a hotel

|  |  |
| --- | --- |
| server in a hotel | Thread |
| server in a hotel – server wont prepare dishes, he will take order and gives it to chef  and chef is the one who prepares dishes | thread – thread also wont execute the line of code  thread will just give those lines to cpu and cpu will execute all those loc  cpu will give the chance to a thread to talk with cpu, in that meeting thread passes all the loc to cpu and he will be executing  all these threads will be competing for the cpu |
| so even if u have 100 bearers In a hotel, but if only 2 chefs are present then its waste  all the bearer/waiter will simply wait untill dishes are prepared by chef  so along with bearers to prepare the dishes chefs also should be more | lly , even if 100 threads are there to execute those lines processor must be free right  1 processor/cpu will give chance to only 1 thread at a time  so in real time along with threads processors also must be present |

|  |  |
| --- | --- |
|  | This means cpu can execute only 1 thread line of content at a time  So os scheduler will keep on switching the threads (few mins thread1, then it will give chance to thread 2 , then thread 3 …) this is called context switching  this is like people waiting for doctor appointment  1 java thread = 1 os thread = platform thread (java thread is a wrapper around os thread)  java thread introduced 25 years ago  so here all the threads will be competing for the cpu |

Many customer will come to hotel to eat food – a server /bearer is the one who will come and take his order and processes his food and he will return the food to cust

In this case the server/bearer is called thread , by default tomcat server will have 200 threads means at a time 200 requests can be handled by tomcat server

All the requests will be assigned to the thread to process

1. Executor service is having a pool of threads == it is like a TCS which consists pool of employees who are on bench

So if any work comes those emp who are in bench will take that work and he will do the project

Facts

Creating threads is costly affair , so in real time u don’t create thread, you should always submit a task to thread pool like executor service thread pool

We cant create a thread pool of 1 million threads- because these are os threads

Advantages

1. If Thread dies suddenly we don’t need to worry, another thread from pool will be assigned to work on our submitted task (like emp exits from tcs- so tcs then gives new resource to client, hence product based companies instead of hiring employees, they will take emp/contractors from tcs/service based companies)
2. We don’t need to create thread object and we don’t need to call start method

Virtual threads

Wht is the problem with platform threads

|  |  |
| --- | --- |
|  | problem here is when order service is calling 3 other services in microservices communication, the main threads/os threads/platform threads will have to wait for those responses   * Most values os threads were in waiting state – this is d problem * Ex:- in reality clients will not keep their most valuable employees idle |

Executor service

A class called

|  |  |
| --- | --- |
| *ExecutorService* es = Executors.*newFixedThreadPool*(20); | //worst as some would have blocked for IO  And we cant increase number of threads based on load |
| Executors.newCachedThreadPool() | Ok—threads are created on demand/workload – but this keeps on increases if more tasks came  Best is Threads will be shutted down automatically if they are not used |
| Executors.*newWorkStealingPool*(3); | Here among 3 threads if any thread is ideal, that thread will steal that work from other threads queue |
| *ScheduledExecutorService* scheduledes = Executors.*newScheduledThreadPool*(2); |  |
| ThreadPoolExecutor | Executes each submitted task using a pooled worker thread |
|  |  |
|  |  |
|  |  |

Key interfaces

Executor, ExecutorService, ScheduledExecutorService, CompletionService,

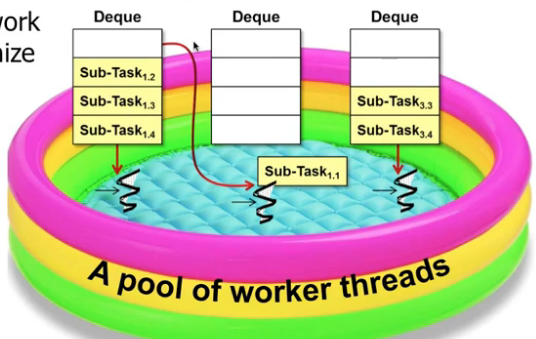
Tasks

These tasks represents the tasks whose instances are executed by another thread

|  |  |  |
| --- | --- | --- |
| Runnable | Callable | Future (I) & impl class is FutureTask(c)  Future (represents result of an async task, like result of callable) |
| Void run() | Object call() | boolean cancel(boolean *mayInterruptIfRunning*);  boolean isCancelled();  boolean isDone();  *V* get() – this is a blocking call & returns the result after some thread executed that task  *V* get(long *timeout*, *TimeUnit unit*)  Callable<String> task = () -> searcher.search(target);  Future<String> future = executor.submit(task); |

dfg

Stealing pool



Here if u see 2nd thread has completed all subtasks, as this is ideal, this will steal another sub tasks which belong to other thread

Here deque- is called double ended queue, means from both sides we can pick the tasks,

Note:- while using this steal pool, its our responsibility to split or fork a main task into multiple subtasks, else other idle threads cant pick those subtasks,

If there is only single task, that task will be owned by a thread, so other threads cant come and pick partially, hence better to split as sub tasks

Note:-

1. so always use thread pools , instead of dynamically creating or spawning a thread per client incurs excessive processing overhead

Fork join pool

Here forking means – splitting into smaller tasks (until u cant split further anymore)

This is based on divide and conquer, it says

If

Problem is small solve it directly

Else

Split into independent parts

Run all sub tasks parallelly (Fork new sub tasks to solve each part)

Join all those sub-tasks

1. split into individual smaller sub tasks (if u split a big task into smaller then only other ideal threads can take this smaller sub tasks,

else only that dedicated thread need to run this big task meantime if other threads are ideal they cant take as it is not a sub task)

1. all these sub tasks will be run in parallel by n threads of same core or diff core processor
2. merge the sub tasks results using join() 🡪 waiting for them to complete or

This is similar to executor service with below diff

Similarities

Here also we will submit the tasks to pool like in executor service

Difference

1. Here work stealing will be there
2. If 20 threads are there for 10 tasks, then each thread will take 1 task, now remaining 10 will be ideal (in fork join pool these will not be ideal if we define those tasks as even more sub tasks), whereas in fork join pool, we should fork means we should divide into even small parallel tasks (not IO operations like db , REST service calls as we can’t split these tasks), now when we split each task into even small tasks

Then remaining ideal threads will tasks these small tasks

When to use this

1. ForkJoinPool may also be appropriate for use with event-style tasks that are never joined.
2. We should not use this IO operations (like db calls, network REST calls)

Misc concepts

Creating a thread in java 21 way

Count down latch

If there are 5 threads, and if u want main thread to wait for those 5 threads completion then we can achieve this In 2 ways

1. Thread.join() – worst and not used in prod
2. Count down latch – internally it will maintain a counter, when any thread completes its exec we should dec the counter, main thread will be blocked at that line it self

Untill count is zero , once count is zero, then main thread will continue its execution